Manipulation des layouts

version 1

Interface Homme-Machine : Unity (Version enseignant)

Voici les objectifs de ce sujet :

— continuer à manipuler l'IDE `Unity` ;

— continuer la création d'un *widget* complexe ;

— exploiter les mécaniques vues précédemment ;

— utiliser les `Layouts`.

1 Description générale du TP

La fois précédente, nous avons réalisé nos premiers widgets complexes en exploitant l'agrégation de plusieurs widgets de base. Nous avons en particulier manipulé le système d'ancre que propose `Unity` pour placer les objets de façon relative.

Ici, nous allons voir une autre méthode un peu plus coûteuse mais offrant une plus grande puissance en terme de placement et qui reprend les points que vous avez étudié dans les années précédentes en IHM : les mises en page (`layout`). La documentation est présente ici : `https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/UIAutoLayout.html`

Solution

Créer un nouveau projet de type `3D Core`.

Composants basiques

— Tester les propriétés du composant `Content Size Fitter` sur un `GameObject` de type `Text` ou `Image`.

Solution

Source : Doc. Content Size Fitter.

Ce `Component` permet de gérer les dimensions du `Component` auquel il est rattaché (pas celui de ses enfants).

- Créer un `Canvas` et y ajouter un `Panel`.
- Dans ce `Panel`, créer un `GameObject Text (Legacy)`.
- Associer à ce `Text` un `Component Content Size Fitter`.
- Tester ses propriétés :
  - `Unconstrained` : on peut redimensionner le `Text` comme on le souhaite ;
  - `Min Size` : fixe les dimensions aux valeurs minimum du `Text` ⇒ attention, ces valeurs valent 0 ;
  - `Preferred Size` : adapte les dimensions du `Text` à son contenu ; tester avec différentes tailles de fonte.

— Faites de même pour le composant `Aspect Ratio Filter`. Pour bien distinguer les effets de chaque composant, il est recommandé d'en associer un seul à la fois au `GameObject`.

Solution

Source : Doc. Aspect Ratio Fitter.

- Retirer le composant `Content Size Fitter` du `Text` précédent.
- Ajouter le composant `Aspect Ratio Fitter` au `Text`.
- Tester ses propriétés :
  - `Width Controls Height` : la hauteur est calculée à partir de la valeur de la largeur, selon le coefficient multiplicateur `Aspect Ratio` (largeur divisée par hauteur) ;
  - `Height Controls Width` : c'est l'inverse

— `Fit In Parent` : le `Text` est agrandi jusqu'à ce que sa largeur ou sa hauteur atteigne celle de son parent (ici, le `Panel`) ; `Aspect Ratio` est toujours modifiable.

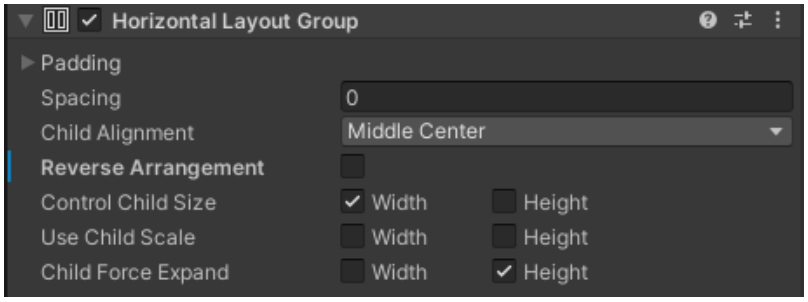
— `Envelope Parent` redimensionne le `Text` jusqu'à recouvrir son parent. Selon `Aspect Ratio`, cela peut entraîner le dépassement du composant par rapport à son parent.

2 Présentation des outils de mise en pages en Unity UI

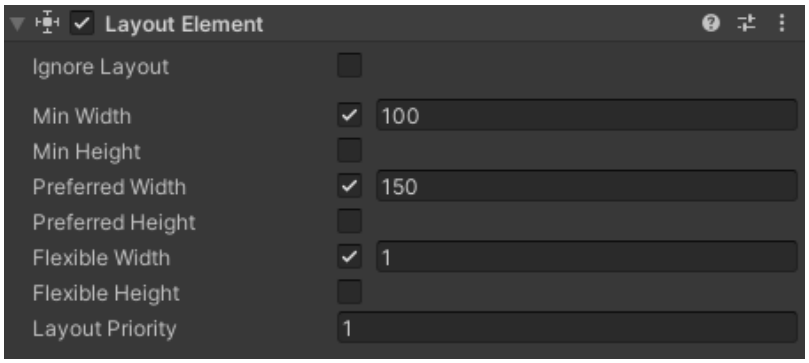
Il est conseillé de lire en détail la documentation citée plus haut pour bien comprendre les aspects et tous les détails. Voici le résumé des points clés :

— Il y a deux types de composants dédiés à la mise en page.

— Les *conteneurs* ou `Layout Group` contrôlent le comportement des widgets fils (enchaînement horizontal, enchaînement vertical ou sous forme de grille...)



— Les composants *éléments* ou `Layout Element` qui indiquent leur présence de mise en page.



Ainsi, chaque conteneur peut avoir des paramétrages différents qui vont faire évoluer les éléments selon les contraintes. Vous ferez attention à certains paramètres qui peuvent forcer le redimensionnement des `widgets` éléments sans les consulter. Vous ferez aussi attention aux éléments qui doivent activer la flexibilité des dimensions voulues pour agrandir dans cette direction (une valeur de 1 peut être suffisante pour indiquer un degré de liberté).

Tests

1. Dans un premier temps, tester les conteneurs `Horizontal Layout Group`, `Vertical Layout Group` et `Grid Layout Group` **sans** ajouter de composant `Layout Element` aux enfants des groupes.

Solution

Horizontal Layout Group Source : Doc. Horizontal Layout Group.

— Supprimer le `Text` précédent et en ajouter un nouveau dans le `Panel`.

— Ajouter un `Slider` dans le `Panel` et le déplacer pour le mettre à côté de `Text`.

— Ajouter au `Panel` le composant `Horizontal Layout Group` : les enfants sont automatiquement replacés dans le `Child Alignment` par défaut : *Upper Left*.

— Modifier les propriétés de ce composant :

— `Padding` : espacement par rapport au bord du `Panel`

— `Spacing` : espacement entre les `GameObject` enfants

- `Child Alignment` (`Upper Left` par défaut) : alignement des enfants
- `Reverse Alignment` : inverse l'ordre des enfants
- `Control Child Size` : modifie les dimensions des enfants en fonction de celles du parent (les propriétés de dimension et de position du `Rect Transform` des enfants sont bloquées).
- `Use Child Scale` : détermine si l'échelle de redimensionnement des enfants est prise en compte dans leur redimensionnement ⇒ tester les effets de cette propriété lorsque les propriétés `Rect Transform > Scale` des enfants sont différentes de 1.
- `Child Force Expand` : force les enfants à se redimensionner pour occuper l'espace utilisable dans le parent.

Vertical Layout Group Source : Doc. Vertical Layout Group.
Même procédure que pour le composant `Horizontal Layout Group`.

Grid Layout Group Source : Doc. Group Layout Group.

- Retirer le `Layout Group` précédent du `Panel`.
- Insérer d'autres widgets dans ce `Panel` et les disposer librement.
- Associer le composant `Grid Layout Group` au `Panel` et modifier ses propriétés
  - `Padding` : cf. plus haut
  - `Cell Size` : définit les dimensions des enfants
  - `Start Corner` : le coin où le premier enfant est placé
  - `Start Axis` : sélectionne l'axe de placement principal (horizontal ou vertical)
  - `Child Alignment` : alignement des enfants s'ils ne remplissent pas tout l'espace disponible
  - `Constraint` : précise le nombre de lignes et de colonnes à respecter. La valeur *Flexible* redispose les enfants selon le redimensionnement du parent.

2. Associer ensuite des `Layout Element` aux enfants des groupes et tester les conteneurs.

Solution

Layout Element Ce composant est ajouté aux enfants eux-mêmes et leurs propriétés se combinent avec celles des `Layout Group` de leur parent. Les propriétés d'un `Layout Element` sont :

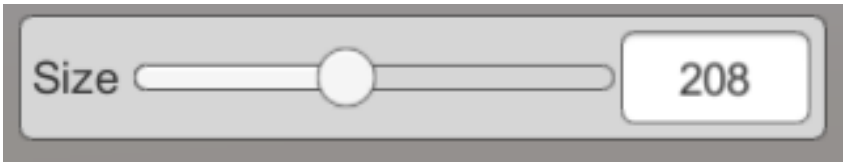
- `Ignore Layout` : le `Layout` est ignoré (pratique si les réglages entrent en contradiction avec ceux d'un parent) ;
- `Min Width, Min Height` : dimensions minimales de cet élément ;
- `Preferred Width, Preferred Height` : dimensions préférées de cet élément. La valeur affectée est calculée en fonction du contenu existant ;
- `Flexible Width, Flexible Height` : valeur relative disponible pour que l'élément courant se redimensionne par rapport à ses frères ;
- `Layout Priority` : si le `GameObject` est associé à plusieurs composants qui disposent chacun de propriétés de Layout, il faut éviter que ces dernière interfèrent. Le composant disposant de la priorité maximale prend le pas sur les autres.

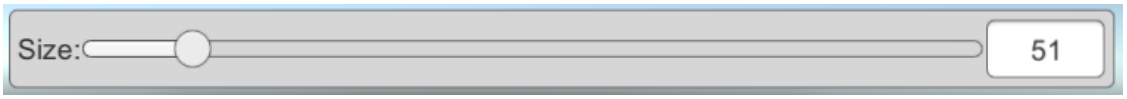
Remarque : certaines propriétés peuvent interférer avec celles du parent. Par exemple, si on utilise un `Grid Layout Group` qui fixe les dimensions des cellules, les propriétés de dimension des éléments de ce groupe devraient être recalculées (voir l'effet avec `Preferred Width/Height` par exemple).

3 Widget : ComplexSlider - le retour en joli

Refaites le widget `ComplexSlider` (soit sous un autre nom, soit après avoir fait une sauvegarde de votre ancien projet/-widget). Pour obtenir un affichage joli, peu importe la largeur que vous donnerez à votre widget, pourvu que le `Slider` prenne la plus grande place.

Veillez à ce que le widget contenant la valeur numérique ne soit pas modifiable directement par l'utilisateur.





Solution

3.1 Création du ComplexSliderJoli

Le principe est de recréer un `Complex Slider` nommé *ComplexSliderJoli* tel que les dimensions du `Text` à gauche du `Slider` et du `Input Field` à droite soient conservées, tandis que le `Slider` au centre est redimensionné en même temps que le `Panel`.

Important : vérifier que le `Canvas` ne possède aucun `Group Layout`, pour éviter qu'il interfère avec les étapes suivantes.

Commencer par insérer un `Panel` dans le `Canvas` avec `UI > Panel`.
Dans ce `Panel`, créer un nouveau `Panel` nommé *ComplexSliderJoli*, insérer un composant `Horizontal Layout Group` et des enfants `Text (Legacy)`, `Slider` et `InputField (Legacy)`. Noter que ces enfants sont automatiquement disposés en haut à gauche du `Panel` (réglage par défaut du Horizontal `Layout Group > ChildAlignment : Upper Left`).

- `Text`
  - modifier les propriétés
    - `Text > Text` : *Size*
    - `Text > Character > Font Size` : 42 (**attention** : cette taille est trop grande pour loger le texte : ce dernier n'apparaît pas actuellement)
    - `Text > Paragraph > Alignment` : *Center / Middle*
  - ajouter le composant `Layout Element` avec les propriétés
    - `Min Width` (cochée) : 160
    - `Preferred Width` (cochée) : 160
- `Slider`
  - modifier les propriétés du composant `Slider`
    - `Min Value` : 0
    - `Max Value` : 999
    - `Whole Numbers` : cocher
  - ajouter le composant `Layout Element` avec les propriétés
    - `Min Width` (cochée) : 100
    - `Flexible Width` (cochée) : 1
- `InputField`
  - propriété `Interactable` : désactiver pour empêcher l'utilisateur d'écrire dans ce champ ;
  - propriété `Transition > Disabled Color` : cette couleur est gris par défaut, passer cette couleur en *blanc* et *alpha = 255* pour suivre l'illustration du sujet ;
  - composant `Text` : modifier les propriétés
    - `Text > Text` : *000* (ou bien directement dans le `Input Field` lui-même),
    - `Text > Character > Font Size` : 36,
    - `Text > Paragraph > Alignment` : *Center / Middle*
  - ajouter le composant `Layout Element` avec les propriétés
    - `Min Width` (cochée) : 160
    - `Preferred Width` (cochée) : 160

Important : C'est cette propriété `Flexible Width` qui active le redimensionnement du `Slider` en même temps que celui du `Panel` parent `ComplexSliderJoli`.

- `ComplexSliderJoli`
  - Propriétés du `Horizontal Layout Group` :
    - `Spacing` : jouer ur cette valeur pour que la poignée du `Slider` ne chevauche pas les autres `widgets` lors de son déplacement aux extrémités.
    - `ChildAlignment` : conserver `Upper Left`
    - `Control Child Size > Height / Width` : valider les deux valeurs pour redimensionner les enfants en fonction de la taille de `ComplexSliderJoli` (plus précisément, la hauteur de tous les enfants sera affectée ; mais seule la largeur du `Text` et du `InputField` ne sera pas modifiée, car la propriété `Flexible Width` de leur composant `Layout Element` n'est pas cochée).
    - `Child Force Expand` : cocher `Height` et décocher `Width` pour adapter la hauteur (pas la largeur) des enfants à celle de leur parent.
  - `Rect Transform` : si la largeur (`Width`) et/ou la hauteur (`Height`) est passée à 0 lors des dernières manipulations, fixer ces valeurs, par exemple *800x600*. En jouant sur ces valeurs (outil `Rect Tool` dans la fenêtre glissante sur la `Scene`), on valide bien le redimensionnement en largeur du `Slider` uniquement.

Remarque : les composants de type `Text` nécessitent une taille minimale pour que leur contenu soit visible. Attention en redimensionnant le parent...

3.2 Script à associer au ComplexSliderJoli

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;

public class ComplexSliderScript : MonoBehaviour
{
	public Slider m_Slider;
	public InputField m_InputField;

	// Start is called before the first frame update
	void Start()
	{
		m_Slider = gameObject.GetComponentInChildren<Slider>();
		m_InputField = gameObject.GetComponentInChildren<InputField>();

		Debug.Log("Slider found: " + m_Slider + " name: " + m_Slider.name);
		Debug.Log("Field found: " + m_InputField + "name: " + m_InputField.name);

		m_Slider.onValueChanged.AddListener(UpdateValueFromFloat);
		m_InputField.onEndEdit.AddListener(UpdateValueFromString);

	}

	// Update is called once per frame
	void Update() { }

	public void UpdateValueFromFloat(float value)
	{
		Debug.Log("float value changed: " + value);
		if (m_InputField) { m_InputField.text = value.ToString(); }
	}

	public void UpdateValueFromString(string value)
	{
		Debug.Log("string value changed: " + value);
		try
		{
			float ff = float.Parse(value);
			if (m_Slider && m_Slider.value != ff) { m_Slider.value = ff; }
		}
		catch(System.Exception e) {
			Debug.Log("error: " + e);
		}
	}
}
```

4 Widget : Spinner

Réalisez le widget du `Spinner` qui consiste à contrôler les évolutions d'un nombre via deux boutons regroupés en bout de ligne.

La valeur de l'incrément (positif ou négatif) sera laissé au choix de l'utilisateur.

Le widget doit être fonctionnel, mais vous pouvez bien sûr changer/adapter selon vos souhaits le coté esthétique du widget. De nouveau, vérifiez que le widget contenant la valeur numérique ne soit pas modifiable directement par l'utilisateur.

Solution

Ce `widget` sera composé de deux parties principales : un `Input Field` et un groupe de deux `Buttons`, disposés de manière à laisser seul l'`Input Field` être redimensionné en fonction des dimensions du parent.

4.1 Création du widget

- Dans le `Canvas` principal, créer un `Panel` nommé *SpinnerPanel*.
- Insérer dans *SpinnerPanel* les enfants suivants :
  - `Input Field (Legacy)`
  - un `GameObject` de type `Empty` nommé *EmptyButtonParent* et dans ce dernier :
    - deux `Buttons (Legacy)` nommés respectivement *ButtonUp* et *ButtonDown*
- Ajouter à *SpinnerPanel* un composant `Horizontal Layout Group` avec les propriétés :
  - `Padding > Spacing` : 35
  - `Child Alignment` : *Upper Left*
  - `Control Child Size` : cocher *Width* et *Height*
  - `Child Force Expand` : cocher *Height*
- Dans `Input Field (Legacy)` :
  - `Composant Input Field`
    - propriété `Interactable` : désactiver pour empêcher l'utilisateur d'écrire dans ce champ
    - `Transition > Disabled Color` : cette couleur est gris par défaut, passer cette couleur en blanc et alpha = 255 pour suivre l'illustration du sujet
    - `Input Field > Text` : 0
  - modifier les enfants
    - `PlaceHolder > Text > Text` : effacer *"Enter text..."*
    - `Text (Legacy) > Character > Font Size` : 42
    - `Text (Legacy) > Paragraph > Alignment` : *Center* et *Middle*
  - ajouter un composant `Layout Element` avec les propriétés
    - `Min Width` : cocher la case, valeur = 100
    - `Preferred Width` : cocher la case, valeur = 100
    - `Flexible Width` : cocher la case, valeur = 1
- Dans `EmptyButtonParent`, ajouter les composants
  - `Vertical Layout Group`
    - `Spacing` : 10
    - `Child Alignment` : *Upper Left*
    - `Control Child Size` : cocher *Width* et *Height*
    - `Child Force Expand` : cocher *Height*
  - `Layout Element` (**ne pas cocher** `Flexible Width` pour laisser au composant `Input field (Legacy)` remplir tout l'espace restant)
    - `Min Width` : cocher la case, valeur = 100
    - `Preferred Width` : cocher la case, valeur = 100
    - `Layout Priority` : 1

  - deux composant de type `Button` respectivement nommés *ButtonUp* et *ButtonDown*
- Dans *ButtonUp*
  - Pour associer une image de flèche
    - récupérer une image de type PNG et dans l'onglet `Project`, clic droit et `Import New Asset...` puis sélectionner le fichier
    - dans `Project`, sélectionner l'image importée et modifier la propriété `Texture Type` : *Sprite (2D and UI)*
    - valider le message de création de l'`Asset`
  - dans le composant `Image`, modifier la propriété
    - `Source Image` en sélectionnant le petit rond à droite du champ et en filtrant la liste avec le nom du fichier
    - si l'image est mal proportionnée par rapport au `Button`, on peut modifier la propriété `Scale` de son composant `Rect Transform`
    - dans l'enfant de type `Text`, modifier la propriété `Text > Text` : effacer le texte par défaut "*Button*"
- Dans *ButtonDown* : répéter les opérations entreprises pour *ButtonUp* en choisissant une autre image.

4.2 Script associé à SpinnerPanel

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;

public class SpinnerScript : MonoBehaviour
/*, IPointerDownHandler, IPointerUpHandler, IDragHandler*/
{
	private Button m_ButtonUp, m_ButtonDown;
	private InputField m_InputField;

	// Increment laisse au choix de l'utilisateur
	public int m_IncrementStep = 1;

	// Start is called before the first frame update
	void Start()
	{
		m_InputField = gameObject.GetComponentInChildren<InputField>();
		if (m_InputField != null) {
			Debug.Log("Input␣Field␣found:␣" + m_InputField + "name:␣" +
			m_InputField.name);
		}

		Button[] buttons = gameObject.GetComponentsInChildren<Button>();
		if (buttons != null) {
			if (buttons[0].name.Equals("ButtonUp") &&
			buttons[1].name.Equals("ButtonDown")) {
				m_ButtonUp = buttons[0];
				m_ButtonDown = buttons[1];
			}
			else if (buttons[1].name.Equals("ButtonUp") &&
			buttons[0].name.Equals("ButtonDown")) {
				m_ButtonUp = buttons[1];
				m_ButtonDown = buttons[0];
			}
			else {
				Debug.Log("Buttons␣Up␣and␣Down␣not␣found!");
			}
		}
		else {
			Debug.Log("Buttons␣not␣found!");
		}

		if (m_ButtonUp != null && m_ButtonDown != null) {
			m_ButtonUp.onClick.AddListener(delegate {
				IncrementField(true);
			});
			m_ButtonDown.onClick.AddListener(delegate {
				IncrementField(false);
			});
		}
	}
```

```
    // Update is called once per frame
        void Update() { }

        private void IncrementField(bool increment) {
                string m_Text = m_InputField.text;
                try {
                        int value = Int32.Parse(m_Text);
                        Debug.Log("value = " + value);
                        if (increment) {
                                value += m_IncrementStep;
                        }
                        else {
                                value -= m_IncrementStep;
                        }
                        Debug.Log("new value = " + value);
                        m_InputField.text = value.ToString();
                }
                catch (FormatException e) {
                        Debug.Log(e.Message);
                }
        }
}
```

5 Aspects avancés sur le système d'événement souris

Nous présentons ici une mécanique pour interagir avec des événements particuliers. Pour cela, consultez le lien suivant qui donne les événements supportés `https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/SupportedEvents.html`. Nous vous proposons un petit exercice sous forme de tutoriel :

- Dans un nouveau projet ou une nouvelle `Scene`[1], ajoutez un `Panel` occupant l'entièreté du `Canvas`.
- Modifiez la couleur du `Panel` pour qu'il soit entièrement transparent.
- Ajoutez un nouveau script.
- Au début du fichier script, ajoutez la ligne

```
using UnityEngine.EventSystems;
```

- Faire hériter le script avec les événements voulus (cf. lien plus haut). Dans notre exemple, nous allons nous concentrer sur les cliques souris et donc nous prendrons l'interface : `IPointerClickHandler`.
- Surchargez les fonctions associées aux interfaces. Ici :

```
public void OnPointerClick(PointerEventData data) { }
```

Solution

5.1 Création d'une scène supplémentaire

Pour ajouter une `Scene` à la *SampleScene* actuelle :

- Menu `File > New Scene > Basic (Built-in)` et cocher `Load additively` pour que la seconde `Scene` apparaisse dans la `Hierarchy`.
- Dans la `Hierarchy`, le nom de la nouvelle `Scene` est *Untitled*. Cliquer sur le symbole "*3 points verticaux*" à droite de ce nom pour sauvegarder cette `Scene`, par exemple sous le nom *MouseScene*.

→ Les fichiers *MouseScene.unity* et *MouseScene.unity.meta* sont créés.

1. Dans ce dernier cas, ajouter dans chaque scène un bouton pour permuter entre les scènes.

— Double-cliquer sur cette `Scene` pour la sélectionner et ajouter un `UI > EventSystem`. **Attention**, si cette sélection n'est pas faite au préalable, ajouter un `EventSystem` n'a pas d'effet : on est renvoyé sur le premier ajouté par défaut dans la `Scene` initiale.

— Si une seule des `Scenes` apparaît dans la `Hierarchy`, déplacer l'autre depuis le `Project` vers la `Hierarchy` pour la faire apparaître.

Important : Dans le menu `File > Build Settings...`, vérifier que toutes les scènes sont bien cochées. Éventuellement, cliquer sur `Add Open Scenes` pour afficher les `Scenes` dans la listes "*Scenes In Build*".

Remarque : les deux `Cameras` ont la même valeur de propriété `Camera > Depth` $= -1$. Dans le mode `Game`, c'est la `Camera` la plus récente (celle de *MouseScene*) qui est affichée par défaut. Pour revenir à la vue sur *SampleScene*, il faut augmenter la valeur de ce champ pour *Main Camera*.
Important : En complément de la remarque précédente, on peut spécifier la `Scene` active en cliquant sur le nom d'une `Scene`, puis `bouton droit > Set Active Scene`.

5.1.1 Peuplement de la nouvelle Scene

— Ajouter un bouton dans *SampleScene* : `GameObject > UI > Legacy > Button (Legacy)`
  — nommer ce bouton *ButtonSwitchToMouseScene*
  — modifier son enfant `Text (Legacy)`
    — Composant `Text`
      — propriété `Text` = *Set MouseScene*
      — propriété `Character > Font Size` = 32
    — Composant `Rect Transform`
      — `ancre` *bottom left*
      — `Pos X = 50 ; Pos Y = 35 ; Pos Z = 0`
      — `Width = 280 ; Height = 45`
      ⇒ au besoin, changer ces valeurs pour ne pas empiéter sur les autres enfants du `Panel`.
— Dupliquer le `Canvas` de *SampleScene*.
— Déplacer ce nouveau `Canvas` en tant qu'enfant de *MouseScene*.
— Dans *MouseScene* :
  — Supprimer si on le souhaite *ComplexSliderJoli* ou *SpinnerPanel*, ou bien les déplacer, pour modifier l'allure du `Canvas`.
  — Renommer le `Button` dupliqué en *ButtonSwitchToSampleScene*.
  — Soit dans *SpinnerPanel*, soit dans un nouveau `Panel`, modifier
    — Composant `Image` : modifier la propriété `Color` pour mettre la valeur du canal alpha à 0 : le `Panel` devient complètement transparent.
  — `Button` *ButtonSwitchToSampleScene*
  — modifier son enfant `Text`
    — Composant `Text` : propriété `Text` = *Set SampleScene*

5.1.2 Script de permutation de Scenes

Associer le script suivant aux `Buttons` *ButtonSwitchToMouseScene* et *ButtonSwitchToSampleScene*.

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;
using UnityEngine.SceneManagement;

// Charge une des scènes du projet.
// Ce script est associé àtous les boutons (un par scène)
// permettant de sélectionner une autre scène

// ATTENTION : un GameObject "EventSystem" doit être placé dans CHAQUE scène
// pour que les iteractions sient prises en compte.

// Source : https://www.youtube.com/watch?v=PpIkrff7bKU
public class SelectSceneScript : MonoBehaviour
{
	private Button m_Button;

	// Start is called before the first frame update
	void Start()
	{
		m_Button = gameObject.GetComponent<Button>();
		if (m_Button == null) {
			Debug.Log("m_Button = null");
		}
		else {
			Debug.Log("m_Button " + m_Button.name + " found.");
		}
		m_Button.onClick.AddListener(delegate {
			SetOtherScene();
		});
	}

	// Update is called once per frame
	void Update() { }

	private void SetOtherScene() {
		Debug.Log("Enter SetOtherScene");
		if (m_Button.name.ToString().Equals("ButtonSwitchToMouseScene")) {
			SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex + 1);
			Debug.Log("Selected button : ButtonSwitchToMouseScene");
		}
		else if (m_Button.name.ToString().Equals("ButtonSwitchToSampleScene")) {
			SceneManager.LoadScene(SceneManager.GetActiveScene().buildIndex - 1);
			Debug.Log("Selected button : ButtonSwitchToSmapleScene");
		}
		else {
			Debug.Log("Error SetOtherScene");
		}
	}
}
```

ATTENTION : parfois, le contenu de la seconde `Scene` dans la `Hierarchy` semble disparaître, mais tout revient dans l'ordre en mode `Game`, quand on permute entre les deux `Scenes` ? ? ?.
On peut ouvrir chaque `Scene` indépendamment dans le menu `File > Open [Recent] Scene`.

5.2 Gestion de la souris

Associer le script suivant dans un Panel ou l'un de ses enfants (peu importe la `Scene`, du moment qu'elle contienne un `EventSystem`).

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;

// Affiche les coordonnees de la souris àchaque clic
public class PanelMouseScript : MonoBehaviour, IPointerClickHandler
{
        public void OnPointerClick(PointerEventData data)
        {
                Debug.Log("OnPointerClick " + data);
        }

}
```

Normalement, les événements gérés par cette mécanique réagissent par défaut et vous pouvez le vérifier avec des messages de `Log`.

A présent, nous souhaitons réaliser les traitements suivants.

1. Lorsque nous cliquons sur une zone de l'écran, nous voulons créer à la volée un `widget` de notre choix à l'écran en tant que fils de notre `Panel` initial (n'oubliez pas qu'un `widget` UI doit avoir comme parent un `Canvas`). On suppose que dans le script, on connaît à l'avance le `widget` qui sera cloné. Pour réaliser cela, consultez la page `https://docs.unity3d.com/ScriptReference/Object.Instantiate.html`.

   Solution

   Dans l'une des `Scenes`, par exemple *MouseScene*, insérer dans le `Canvas > Panel` un `widget` quelconque, par exemple un `Button` nommé *CloneWidget*.

   Puis associer le script suivant au `Canvas > Panel`.

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;

// Script associe a un Panel
// Clone un widget choisi par l'utilisateur lors d'un clic de souris.
// Ce widget sera integre comme fils du Panel.
// La position du clone est determinee par la position du curseur de la souris
public class CloneWidgetScript : MonoBehaviour, IPointerClickHandler
{
	private static string TAG = "CW - ";
	private Camera m_Camera;
	public GameObject m_Widget; // Widget selectionne par l'utilisateur

	void Start() {
		// https://docs.unity3d.com/ScriptReference/Camera.html
		// The first enabled Camera component that is tagged "MainCamera"
		//  (Read Only).
		m_Camera = Camera.main;
		if (m_Camera == null) { Debug.Log(TAG + "Camera not found !"); }

		Button[] buttons = gameObject.GetComponentsInChildren<Button>();
		if (buttons != null) {
			int i = 0;
			while (i < buttons.Length &&
			(! buttons[i].name.Equals("ButtonToClone"))) { i++; }

			if (i == buttons.Length) {
				Debug.Log(TAG + "Button to clone not found!");
				m_Widget = null;
			}
			else {
				m_Widget = buttons[i].gameObject;
				Debug.Log(TAG + m_Widget.name + " found!");
			}
		}
	}

	public void OnPointerClick(PointerEventData data)
	{
		Debug.Log(TAG + "OnPointerClick " + data);

		if (m_Widget != null) {
			Vector2 localPoint = new Vector2(data.position.x, data.position.y);
			Debug.Log(TAG + "Local point: " + localPoint);

			GameObject obj = Instantiate(m_Widget,
			localPoint,
			Quaternion.identity,
			gameObject.transform);
		}
		else {
			Debug.Log(TAG + "Pas de widget selectionne !");
		}
	}
}
```

2. Enfin, vous êtes prêt à créer un script `ResizeWidget` qui consiste à agrandir en largeur/hauteur un `widget` quelconque en faisant un drag sur ce `widget`.

Solution

On associe le script suivant au dernier `Button` créé dans *MouseScene* : *ButtonToClone* (mais on pourrait utiliser n'importe quel widget).

```
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.EventSystems;

// Redimensionne le Widget associe a ce script, selon le "drag" de souris
// Les directions "Droite" et "Bas" permet de grossir le widget
public class ResizeWidgetScript : MonoBehaviour, IPointerDownHandler,
   IPointerUpHandler, IDragHandler
{
        private Vector2 previousPointerPosition;
        private Vector2 currentPointerPosition;
        private RectTransform rectTransform;
        private static string TAG = "RW - ";

        // Use Awake to initialize variables or states before the application starts.
        private void Awake() { rectTransform = GetComponent<RectTransform>(); }

        public void OnPointerDown(PointerEventData data) {
                Debug.Log(TAG + "OnPointerDown " + data);
        }

        public void OnPointerUp(PointerEventData eventData) {
                Debug.Log(TAG + "OnPointerUp " + eventData);
        }

        public void OnDrag(PointerEventData data) {
                Debug.Log(TAG + "OnDrag " + data);
                rectTransform = GetComponent<RectTransform>();

                if (rectTransform == null) {
                        Debug.Log(TAG + "RectTransform not found !");
                        return;
                }

// https://docs.unity3d.com/ScriptReference/RectTransform-sizeDelta.html
                Vector2 sizeDelta = rectTransform.sizeDelta;
                Debug.Log(TAG + "Current Size: " + sizeDelta);

// https://docs.unity3d.com/ScriptReference/
RectTransformUtility.ScreenPointToLocalPointInRectangle.html
                RectTransformUtility.
                ScreenPointToLocalPointInRectangle(rectTransform,
                        data.position,
                        data.pressEventCamera,
                        out currentPointerPosition);
                Vector2 resizeValue = currentPointerPosition - previousPointerPosition;

                Debug.Log(TAG + "Resize: " + resizeValue);

                sizeDelta += new Vector2(resizeValue.x, -resizeValue.y);

                rectTransform.sizeDelta = sizeDelta;
                Debug.Log(TAG + "New size: " + rectTransform);

                previousPointerPosition = currentPointerPosition;
        }
}
```